



**Hewlett Packard
Enterprise**

CHAPEL 1.23 RELEASE NOTES: COMPILER AND TOOL IMPROVEMENTS

Chapel Team

October 15, 2020

OUTLINE

- [Error Message Improvements](#)
- [Mason Improvements](#)
- [Supporting Protocol Buffers](#)

ERROR MESSAGE IMPROVEMENTS

A scenic landscape photograph featuring a calm lake in the foreground that perfectly reflects the surrounding environment. A dense line of trees with vibrant yellow autumn foliage stands on the right side of the lake. In the background, a range of rugged mountains is visible, with the most prominent peak being a large, craggy mountain covered in patches of snow. The sky is filled with dark, heavy clouds, suggesting an overcast or stormy day. The overall mood is serene yet dramatic.

ERROR MESSAGE IMPROVEMENTS

Background

- Previously, error messages in generic functions could be hard to follow

```
1 proc h(arg) { arg = 11; }  
2 proc g(arg) { h(arg); }  
3 proc f(arg) { g(arg); }  
4 f("hi");
```

```
prog.chpl:1: In function 'h':  
prog.chpl:1: error: Cannot assign to string from int(64)  
prog.chpl:2: Function 'h' instantiated as: h(arg: string)
```

- This error could be more helpful
 - In this case, 'h(string)' should probably never be called
 - Could the error show what called 'h(string)' ?



ERROR MESSAGE IMPROVEMENTS

This Effort

- Extended many error messages to include a callstack for more context:

```
1 proc h(arg) { arg = 11; }  
2 proc g(arg) { h(arg); }  
3 proc f(arg) { g(arg); }  
4 f("hi");
```

```
prog.chpl:1: In function 'h':
```

```
prog.chpl:1: error: Cannot assign to string from int(64)
```

```
  prog.chpl:2: called as h(arg: string) from function 'g'
```

```
  prog.chpl:3: called as g(arg: string) from function 'f'
```

```
  prog.chpl:4: called as f(arg: string)
```

```
note: generic instantiations are underlined in the above callstack
```

- Errors now include bold font and underlining when outputting to a supported terminal
- ‘--print-callstack-on-error’ available to request callstack for concrete functions as well as generic



ERROR MESSAGE IMPROVEMENTS

Next Steps

- Print out regions of code with pointers similar to other compilers
- Consider making use of bold font and underlining in other error messages
- Consider adding color output



MASON IMPROVEMENTS

A scenic landscape photograph featuring a range of rugged, snow-dusted mountains under a dramatic, cloudy sky. In the foreground, a dense forest of trees with vibrant yellow and orange autumn foliage stands on a grassy bank. A calm body of water in the lower half of the image perfectly reflects the mountains, trees, and sky, creating a symmetrical effect. The overall mood is serene and majestic.

MASON IMPROVEMENTS

Background and This Effort

Background:

- Mason is Chapel's package manager and build tool
- Users can publish packages to chapel-lang/mason-registry on GitHub to make them accessible to other users
 - they can also publish to their own internal or public registries for sharing packages

This Effort:

- Added several features and quality-of-life improvements to mason
- Implemented as a Google Summer of Code project
 - Student: Ankush Bhardwaj
 - Mentors: Ben Albrecht, Sam Partee, and Krishna Kumar Dey (Chapel GSoC 2019 Alum)



MASON IMPROVEMENTS

Interactive Package Creation

Background: Users could create mason packages with ‘mason new’ or ‘mason init’

- ‘mason new Foo’ created a bare-bones package named ‘Foo’
- ‘mason init’ created a bare-bones package in the current directory, preserving any existing package contents
- Package metadata such as name, version, and Chapel versions could be edited in ‘Mason.toml’ after creation

This Effort: Implemented interactive ‘mason new’ and ‘mason init’ commands

- ‘mason new’ with no arguments begins an interactive session where the user supplies its metadata
- ‘mason init’ begins an interactive session
 - ‘mason init --default’ disables interactive mode and fills all fields with their default values

Impact: Creating a mason package is now more user-friendly



MASON IMPROVEMENTS

Interactive Package Creation Example

> **mason new**

...

Package name : **Foo**

Package version (0.1.0):

Chapel version (1.23.0):

License (None): **Apache-2.0**

[brick]

name = "Foo"

version = "0.1.0"

chplVersion = "1.23.0"

license = "Apache-2.0"

[dependencies]

Is this okay ? (Y/N): **y**

Created new library project: Foo



MASON IMPROVEMENTS

Automatic Registry Creation

Background: Using a local mason package requires “publishing” it to a local registry

- Creating the registry can be laborious for users and is prone to user errors in terms of directory structure

This Effort: Implemented ‘mason publish --create-registry’ to create a registry automatically

- ‘mason publish --create-registry <path>’ creates a local registry in the designated path
- Users can then publish their locally developed packages to this registry with: ‘mason publish <path>’

Impact: Creating a local registry and using local packages is easier

```
> mason publish --create-registry ~/my-registry
Initialised local registry at /Users/foo/my-registry
...
```

In a mason package

```
> mason publish ~/my-registry
Successfully published package to /Users/foo/my-registry
```



MASON IMPROVEMENTS

Bash Completion

Background: Bash completion was supported for the Chapel compiler but not for mason

This Effort: Implemented bash completion for mason

- Completion commands are generated by parsing the 'mason --help' output to increase maintainability

Impact: Mason is now easier to use

```
> source util/devel/mason-completion.bash
> mason <tab>
add          clean      env          init         publish      run          system      update
build       doc          external    new          rm           search      test
> mason build --<tab>
--example    --force      --help      --no-update  --release    --savec     --show
--update
> mason external <tab>
compiler    find         info         install      search       uninstall
```



MASON IMPROVEMENTS

Manifest License Field

Background: Mason packages use a manifest file to specify package metadata

- License feature was not yet officially supported in 'Mason.toml' manifest file

This Effort: Added a required license field to mason packages

- Uses license identifiers from Software Package Data Exchange (SPDX) license list
- Field is required, but 'none' is a valid value if no license has been chosen yet
- mason-registry CI tests will reject any package without a license field or with an invalid license value
- Packages published prior to this feature are not impacted, but new versions will require a license field

Impact: Users can now specify a license in their mason packages

```
> cat Mason.toml
[brick]
name = "Foo"
version = "0.1.0"
chplVersion = "1.23.0"
license = "Apache-2.0"
```



MASON IMPROVEMENTS

Registry CI Testing Improvements

Background: Chapel's mason registry runs several checks within CI testing on mason-registry PRs

- Checks included:
 - verifying existence of manifest file: 'Mason.toml'
 - verifying existence of package source file: 'src/<package>.chpl'

This Effort: Added more CI testing checks

- Checks now include:
 - verifying that all required manifest fields are present and valid
 - validating the license with SPDX
 - validating the version is formatted correctly in git tag
 - checking for namespace collisions with other packages

Impact: Verifying correctness of published packages is more automated



MASON IMPROVEMENTS

Impact and Next Steps

Impact: Several features and quality-of-life improvements have been added to mason

Next Steps:

- Convert Chapel's package modules (and possibly standard modules) to mason packages
- Formalize and document package acceptance criteria for chapel-lang/mason-registry
- Add 'mason bench' for benchmarking Chapel programs
- Address other known bug fixes and feature requests





SUPPORTING PROTOCOL BUFFERS

SUPPORTING PROTOCOL BUFFERS

Background and This Effort

Background:

- [Protocol Buffers](#) are a language-neutral, platform-neutral, extensible mechanism for serializing structured data
- The protocol buffer language supports specifying the schema for structured data
- This schema is compiled into language-specific bindings
- The protobuf compiler ‘protoc’ uses *plugins* to generate code for many languages
- Protocol Buffers can enable many interoperability scenarios

This Effort: Implement Protocol Buffer support in Chapel

- Developed a ‘protoc’ plugin to generate Chapel code
- Developed a module containing runtime support for encoding and decoding Protocol Buffer formats
- Implemented as a Google Summer of Code project
 - Student: Aniket Mathur
 - Mentors: Audrey Pratt, Michael Ferguson, Lydia Duncan



SUPPORTING PROTOCOL BUFFERS

This Effort

// addressbook.proto

```
syntax = "proto3";  
package addressbook;  
message Person {  
  string name = 1; // 1 here is a field number  
  int32 id = 2; // Unique ID number for this person.  
  string email = 3;  
  enum PhoneType {  
    MOBILE = 0; HOME = 1; WORK = 2;  
  }  
  message PhoneNumber {  
    string number = 1;  
    PhoneType phntype = 2;  
  }  
  repeated PhoneNumber phones = 4;  
}
```

protoc

// dst/addressbook.chpl

```
/*  
  Generated by the protocol buffer compiler. DO NOT EDIT!  
  source: addressbook.proto  
*/  
  
module addressbook {  
  use ProtobufProtocolSupport;  
  use List;  
  use Map;  
  
  // Messages  
  record Person {  
    // Fields  
    var name: string; ...  
    proc serialize(ch) throws {...}  
    proc deserialize(ch) throws {...}  
  }  
}
```

SUPPORTING PROTOCOL BUFFERS

This Effort

```
# compile a .proto file to a Chapel module
protoc --chpl_out=dst addressbook.proto

// addressProgram.chpl
use addressbook; // use the generated message module
use IO;

// create a message
var messageObj: Person;
messageObj.name = "John";
var phoneNumber: Person_PhoneNumber;
phoneNumber.number = "555-4321";
phoneNumber.phntype = Person_PhoneType.HOME;
messageObj.phones.append(phoneNumber);

// output message to file
var file = open("out", iomode.cw);
var writingChannel = file.writer()
messageObj.serialize(writingChannel);
```

SUPPORTING PROTOCOL BUFFERS

Impact and Next Steps

Impact: Chapel users can now easily use a powerful interoperability tool

Next Steps: Improve based upon user feedback





OTHER COMPILER AND TOOL IMPROVEMENTS

OTHER COMPILER AND TOOL IMPROVEMENTS

For a more complete list of compiler and tool improvements in the 1.23 release, refer to the following sections in the [CHANGES.md](#) file:

- ‘Mason Improvements’
- ‘New Tools/ Tool Changes’
- ‘Error Messages / Semantic Checks’





THANK YOU

<https://chapel-lang.org>
@ChapelLanguage

